CHAPTER 6

OPTIMIZATION TECHNIQUES FOR PERFORMANCE IMPROVEMENT
6.1 Introduction

The handwritten character recognition system for Marathi script using neural networks was studied in the previous chapter. After going through the structural classification, the results obtained were encouraging. This further motivated us to investigate the methods for improving the results. But, the characters which got classified to more number of structural classes due to large shape variations showed lesser recognition rate, thereby reducing the overall performance. Thus, the recognition rates can be further improved by increasing the samples in the database, selecting the optimum techniques and parameters at the feature extraction and recognition stages. The network parameters such as activation function, number of neurons in the hidden layer, the network training function etc. can be varied and selected for the optimum performance of the network. This chapter discusses the various techniques adopted towards improvement of the result at various stages of the proposed system. Section 6.2 discusses the optimization at data collection stage. Section 6.3 presents the optimization at feature extraction stage. Section 6.4 describes the optimization carried out in the neural network classifier stage. Section 6.5 discusses the results obtained from the optimized system while Section 6.6 discusses the concluding remarks.

6.2 Optimization at data collection stage

The characters in the database are increased. This is done by increasing the number of writers and the number of samples per character. More than 50 writers with more than 400 samples per character are collected. This results to more than 16,000 samples in the database. 60% of these samples are used for training, 20% is used for validation and 20% is used for testing.

6.3 Optimization at feature extraction stage

During structural classifications, it is observed that some characters as shown in Figure 6.1 show very less variation in their shape. Thus they get classified to very few structural classes. Also, their shape is not too complex which gives good classification
results without much effort devoted towards development of sophisticated feature extraction techniques.

![Image of characters]

Figure 6.1 Characters with lesser shape variation

Some character pairs in Marathi script show similarity in their shape and even a slight variation in the writing style can easily misclassify the character to the other one. Some of such pairs are shown in Figure 6.2. These characters pose difficulties in recognition and demand an efficient feature extraction tool.

![Table of similar characters]

Figure 6.2 Similar characters in Marathi script

![Image of characters]

Figure 6.3 Characters with complex shape

Some characters as shown in Figure 6.3 are very complex. A slight variation in the writing style classifies them to different structural classes. As a result, the number of samples per character in a structural class reduces. They too demand an efficient feature
extraction technique. All these reasons demand a robust feature extraction technique that handles large number of characters in a class with large variations in their writing style, also classifies similar characters at the same time. Some feature extraction techniques were studied in the previous chapter. The feature extraction techniques which can extract more meaningful features are discussed next.

6.3.1 Proposed wavelet approximation features

The wavelet transform exhibits the features like separability, scalability, translatability, orthogonality and multiresolution capability. The discrete wavelet transform of an image \( f(x,y) \) of size \( M \times N \) is

\[
W_\psi(j_0,m,n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \phi_{j_0,m,n}(x,y)
\]  

(6.1)

\[
W'_\psi(j,m,n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \psi_{j,m,n}^j(x,y)
\]

(6.2)

where,

\[
\phi_{j,m,n}(x,y) = 2^{j/2} \phi(2^j x - m, 2^j y - n)
\]

(6.3)

and

\[
\psi_{j,m,n}^j(x,y) = 2^{j/2} \psi^j(2^j x - m, 2^j y - n)
\]

(6.4)

are the two dimensional scaling and wavelet functions respectively and the index ‘i’ identifies the directional wavelets that takes the values H, V and D i.e. horizontal, vertical and diagonal details respectively. \( j_0 \) an arbitrary starting scale and the \( W_\psi(j_0,m,n) \) coefficients define an approximation of \( f(x,y) \) at scale \( j_0 \). The \( W'_\psi(j,m,n) \) coefficients add horizontal, vertical and diagonal details for scales \( j \geq j_0 \). Normally, \( j_0 = 0 \) \( N = M = 2^j \) so that \( j = 0, 1, 2, \ldots, J-1 \) and \( m, n = 0, 1, 2, \ldots, 2^j-1 \). The decomposition is done with respect to Daubechies wavelet. Here, \( j_0=0 \) and \( M=N=16 \) which is equal to \( 2^4 \), hence \( j=0, 1, 2, \ldots, 7 \) and \( m, n=0, 1, 2, \ldots, 15 \).
The discrete wavelet transform can be implemented using digital filters and down samplers. The high pass or detail component characterizes the image’s high-frequency information with vertical orientation; the low-pass, approximation component contains its low-frequency, vertical information. Both sub images are then filtered column wise and down sampled to yield four quarter size output images. These sub images are shown in Figure 6.4. The binary character is resized to a fixed size of 16x16 after two stage structural classification. The single level decomposition generates 8x8 approximation coefficients along with same sized vertical, horizontal and diagonal details. The 64 approximation coefficients obtained are used as features and applied as the input to the neural network.

### 6.3.2 Proposed modified wavelet approximation features

The separation between the features of the characters can be further increased by convolution operation thus resulting into increased recognition rate. The modified wavelet features are obtained by convolving the approximation features with themselves.

Convolution between a 2D signal \( f(x, y) \) and a template \( g(x, y) \) is defined as,

\[
 f * g = \sum_{(x,y) \in T} f(x', y')g(i-x', j-y') \tag{6.5}
\]

where, \( x' = x + i \) and \( y' = y + j \).

The operation of convolution of approximate features with themselves is equivalent to generation of mother wavelet for each character. The image of size 15x15

![Figure 6.4 Single level 2D Wavelet decomposition](image-url)
is obtained after convolving the 8x8 approximation features with themselves after a 2D single level decomposition. The detail features are not used as they loose the information to some extent. The 15x15=225 features are down sampled to a feature vector of 75 features. The use of a separate wavelet for each character generates a separate 2D feature space for each character. It implies that we are having a different co-ordinate system for each character which increases the distance between the two characters thereby reducing the misclassification. The modified wavelet features are also applied to the neural network for recognition.

6.4 Optimization at recognition stage

The multilayer perceptron architecture used in the proposed system is studied in the previous chapter. The performance of this neural network can be improved by proper selection of network parameters. The various parameters that can affect the performance are as follow:

- Number of hidden layers
- Number of neurons in the hidden layer
- Activation functions
- Training algorithm

The next section discusses the selection of following parameters in detail.

6.4.1 Selection of number of hidden layers

For nearly all problems, one hidden layer is sufficient. Using two hidden layers rarely improves the model, and it may introduce a greater risk of converging to a local minima. There is no theoretical reason for using more than two hidden layers. Hence one hidden layer is maintained in the design.

6.4.2 Selection of hidden neurons

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final
output. Both the number of hidden layers and the number of neurons in each of these hidden layers must be carefully considered. Using too few neurons in the hidden layers will result in something called under fitting. Under fitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set. Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in over fitting. Over fitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

The best number of hidden units depends in a complex way on:

- The numbers of input and output units
- The number of training cases
- The amount of noise in the targets
- The complexity of the function or classification to be learned
- The type of hidden unit activation function

There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.
- These three rules provide a starting point. Ultimately, the selection of neurons in the hidden layer comes down to trial and error.

In the proposed system, the number of hidden neurons was selected equal to the number of output neurons initially. Then they were selected equal to the number of
inputs. But this required more memory and sometimes resulted in out-of-memory error when number of features applied was large. Hence a rule-of-thumb was used which decided the number of hidden neurons for all the 24 neural networks built for each structural class. The rule used is square root of the product of input and output neurons.

### 6.4.3 Selection of hidden layer activation function

Initially the activation function used for the hidden layer was log sigmoid (logsig) with the range 0 to 1. On changing this to hyperbolic tangent sigmoid (tansig) activation function with the range from -1 to +1, the improvements in the recognition accuracies is seen. The activation functions are shown in Figure 6.5.

![Activation functions](image)

**Figure 6.5 Activation functions**

### 6.4.4 Selection of training algorithm

Once the network weights and biases are initialized, the network is ready for training. The training process requires a set of examples of proper network behavior, network inputs $p$ and target outputs $t$. During training the weights and biases of the network are iteratively adjusted to minimize the network performance function. The performance function for feed forward networks is mean square error (mse), the average squared error between the network outputs $a$ and the target outputs $t$.

The different training algorithms for feed forward networks use the gradient of the performance function to determine how to adjust the weights to minimize the performance. The gradient is determined using a technique called back propagation, which involves performing computations backward through the network. In the basic back propagation training algorithm, the weights are moved in the direction of the negative gradient.
There are two different ways in which this gradient descent algorithm can be implemented: incremental mode and batch mode. In incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In batch mode, all the inputs are applied to the network before the weights are updated.

**Batch Gradient Descent (GD)**

In the batch steepest descent training function, the weights and biases are updated in the direction of the negative gradient of the performance function. There is only one training function associated with a given network. The learning rate (lr) is multiplied by the negative of the gradient to determine the changes in the weights and biases. The larger the learning rate, the bigger the step. If the learning rate is made too large, the algorithm becomes unstable. If the learning rate is set too small, the algorithm takes a long time to converge.

**Gradient Descent with Momentum (GDM)**

Gradient descent with momentum, allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network can get stuck in a shallow local minimum. With momentum a network can slide through such a minimum. Gradient descent with momentum depends on two training parameters. The parameter ‘lr’ indicates the learning rate, similar to the simple gradient descent and the parameter ‘mc’ is the momentum constant that defines the amount of momentum. Usually ‘mc’ is set between 0 (no momentum) and values close to 1 (lots of momentum). A momentum constant of 1 results in a network that is completely insensitive to the local gradient and, therefore, does not learn properly.)

**Faster Training algorithms**

The previous section presented two back propagation training algorithms: gradient descent, and gradient descent with momentum. These two methods are often too slow for practical problems. This section discusses several high-performance algorithms that can converge from ten to one hundred times faster than the algorithms discussed previously. All the algorithms in this section operate in batch mode.
These faster algorithms are:

- Conjugate gradient (CG)
- Quasi-Newton (QN)
- Levenberg-Marquardt (LM)

**Conjugate Gradient Algorithm**

The basic back propagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient), the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions. The conjugate gradient algorithms require a line search at each iteration. This line search is computationally expensive, because it requires that the network response to all training inputs be computed several times for each search. The scaled conjugate gradient algorithm (SCG), developed by Moller, is designed to avoid the time-consuming line search.

**One Step Secant Algorithm**

The one step secant (OSS) method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms. This algorithm does not store the complete Hessian matrix; it assumes that at each iteration, the previous Hessian was the identity matrix. It requires slightly more storage and computation per epoch than the conjugate gradient algorithms. It can be considered a compromise between full quasi-Newton algorithms and conjugate gradient algorithms.

**Levenberg-Marquardt algorithm**

Like the quasi-Newton methods, the Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix. The training of the network using this algorithm is explained in the following steps:
Step 1: The sum of squared errors over all inputs, $F(x)$ is calculated after applying the inputs to the first layer. The performance index for multilayer network training is the mean squared error. If each target occurs with equal probability, the mean squared error is proportional to the sum of squared errors over the $Q$ targets in the training set.

$$F(x) = \sum_{q=1}^{Q} (t_q - a_q)^T (t_q - a_q)$$ \hspace{1cm} (6.6)

This equation is equivalent to the performance index.

Step 2: Next the Jacobian matrix is calculated by,

$$J(x) = \begin{bmatrix}
\frac{\partial e_{1,1}}{\partial w_{1,1}} & \frac{\partial e_{1,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w_{1,r}} & \frac{\partial e_{1,1}}{\partial b_1} \\
\frac{\partial e_{2,1}}{\partial w_{1,1}} & \frac{\partial e_{2,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w_{1,r}} & \frac{\partial e_{2,1}}{\partial b_1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\frac{\partial e_{m,1}}{\partial w_{1,1}} & \frac{\partial e_{m,1}}{\partial w_{1,2}} & \cdots & \frac{\partial e_{m,1}}{\partial w_{1,r}} & \frac{\partial e_{m,1}}{\partial b_1}
\end{bmatrix}$$

(6.7)

The sensitivities with the recurrence relations are calculated after initializing with

$$\tilde{S}_q^M = -\dot{F}^M (n_q^M)$$ \hspace{1cm} (6.8)

$$\dot{F}^M (n_q^M) = \begin{bmatrix}
\dot{f}^m (n_1^n) & 0 & \cdots & 0 \\
0 & \dot{f}^m (n_2^n) & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \dot{f}^m (n_q^n)
\end{bmatrix}$$ \hspace{1cm} (6.9)

where Each column of the matrix $\tilde{S}_q^M$ must be back propagated through the network to produce one row of the Jacobian matrix using the equation

$$s^n = \dot{F}^m (n_q^n)(W^{m+1})^T s^{m+1}$$ \hspace{1cm} (6.10)

The columns can also be propagated together using
The total Marquardt sensitivity matrices for each layer are then created by augmenting the matrices computed on each input

\[ \tilde{S}^m = \begin{bmatrix} \tilde{S}_1^m \\ \vdots \\ \tilde{S}_q^m \end{bmatrix} \]  

The elements of the Jacobian matrix are then computed by

\[ [J]_{kj} = \frac{\partial e_k}{\partial x_j} = \frac{\partial e_{i,k}}{\partial w_{i,j}} = \tilde{S}_{i,k}^m \times a_{i,j}^{m-1} \]  

Or if \( x_j \) is a bias,

\[ [J]_{kj} = \frac{\partial e_k}{\partial x_j} = \frac{\partial e_{i,k}}{\partial b_{i,j}^m} = \tilde{S}_{i,k}^m \]  

Step 3: \( \Delta x_k \) is obtained by solving,

\[ \Delta x_k = -J^T(x_k)J(x_k) + \mu_k I \]  

Step 4: The sum of squared errors are recomputed using \( x_k + \Delta x_k \). If this new sum of squares is smaller than that computed in Step 1, then divide \( \mu \) by \( \vartheta \), let \( x_{k+1} = x_k + \Delta x_k \) (\( \mu \) and \( \vartheta \) are the Marquardt parameters, \( \mu \) set to some smaller value e.g. 0.01 and \( \vartheta > 1 \)) and go back to Step 1. If the sum of squares is not reduced, then multiply \( \mu \) by \( \vartheta \) and go back to Step 3. The algorithm is assumed to have converged when the norm of the gradient \( \nabla F(x) = 2J^T(x)v(x) \), where \( J(x) \) is the Jacobian matrix is less than some predetermined value, or when the sum of squares has been reduced to some error goal.

The Levenberg-Marquardt algorithm is the fastest back propagation algorithm. The key drawback of the Levenberg-Marquardt algorithm is the storage requirement. The algorithm must store \( n \times n \) matrix where, \( n \) is the number of parameters (weights and biases) in the network. When the number of parameters is large, it may be impractical to use the Levenberg-Marquardt algorithm due to memory requirement. This limits the number of inputs to the neural network.

### 6.5 Experiments and results

The training samples for training the networks are increased. The feature extraction techniques using wavelet transform are implemented. The approximation
wavelet features are calculated using single level wavelet decomposition. The features are further separated in the feature space by convolving the wavelet approximation coefficients with themselves. Figures 6.6 and 6.7 shows the Euclidean features (Refer section 5.4.6) for the characters ‘cha’ i.e. चा and ‘gha’ i.e. घा in vertical and horizontal directions respectively for 16x16 resize factor. The figures show that the features are quite similar inspite of characters being different. The character ‘cha’ is misclassified as ‘gha’ when Euclidean features are used.

![Character 'cha'](image1.png)

![Character 'gha'](image2.png)

Figure 6.6 Euclidean features for character ‘cha’

Figure 6.7 Euclidean features for character ‘gha’
Figure 6.8 Wavelet approximation features for character ‘cha’

Figure 6.9 Wavelet approximation features for character ‘gha’

Figure 6.10 Modified Wavelet approximation features for character ‘cha’

Figure 6.11 Modified Wavelet approximation features for character ‘gha’
Figures 6.8 and 6.9 show the wavelet approximation features in the vertical and horizontal directions respectively for the same characters. There is a significant difference in the features of both the characters. These features when applied to the neural network correctly classify the character ‘cha’. Even the modified approximation features as shown in Figures 6.10 and 6.11 correctly classify the character ‘cha’. The x-axis shows the increased range obtained by convolution operation.

Another example shows how the modified wavelet approximation is more efficient than the wavelet approximation features. Figures 6.12 and 6.13 show the Euclidean distance features for the characters ‘ya’ i.e. या and ‘tha’ i.e. ठा in vertical and horizontal directions respectively. The characters are almost similar and the classification is quite difficult.
The wavelet approximation features are presented in Figures 6.14 and 6.15 respectively and the modified wavelet approximation features are presented in Figures 6.16 and 6.17 respectively. Results show that both the Euclidean features and the wavelet approximation features misclassified the character ‘ya’ as ‘tha’. While the modified wavelet approximation features could classify the characters correctly.
After analyzing the effect of each parameter on the recognition result, the neural network parameters are set as shown in Table 6.1.

Table 6.1 Neural network parameter settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of inputs</td>
<td>With Wavelet approximation features: 64</td>
</tr>
<tr>
<td></td>
<td>With Modified wavelet approximation features: 75</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>1</td>
</tr>
<tr>
<td>Number of neurons in hidden layer</td>
<td>Equal to the square root of the product of number of inputs and number of outputs</td>
</tr>
<tr>
<td>Hidden layer activation function</td>
<td>hyperbolic tangent sigmoid transfer function</td>
</tr>
<tr>
<td>Number of neurons in output layer</td>
<td>Number of characters in the structural class</td>
</tr>
<tr>
<td>Output layer activation function</td>
<td>Linear</td>
</tr>
<tr>
<td>Goal</td>
<td>0.001</td>
</tr>
<tr>
<td>Error function</td>
<td>mse</td>
</tr>
<tr>
<td>Maximum number of epoch</td>
<td>300</td>
</tr>
<tr>
<td>Training algorithm</td>
<td>Levenberg-Marquardt algorithm</td>
</tr>
</tbody>
</table>

All the 24 neural networks built for each structural class are trained for the increased dataset as per the parameters given above. Both the feature extraction methods are applied and the results were analyzed.
Figure 6.18 Training graphs for 64-18-5 network a) GD (lr=0.5), b) GDM (lr=0.5, m=0.5), c) QN, d) CG, & e) LM

The performance graphs for training the MBE/II class with 590 characters with modified wavelet approximation features using various training functions like GD, GDM, QN, CG and LM (discussed in Section 6.4.4) respectively. The training graphs in
Figure 6.18 show performance of the neural network designed with the specified parameters. The graphs show that the Levenberg-Marquardt algorithm is the fastest algorithm which converges in just 5 epochs. Hence this training function was chosen for all the networks. Moreover, the misclassified characters with this function closely match the actual character. This can help in post-processing e.g. lexicon, to recognize the correct character in the word. The training time (in seconds) and the number of epoch for training MBE/11 class using various training functions with different features are given in Table 6.2.

### Table 6.2 Training performance with different features

<table>
<thead>
<tr>
<th>Features</th>
<th>GD</th>
<th>QN</th>
<th>CG</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Epochs</td>
<td>Tr. Time (s)</td>
<td>Epochs</td>
<td>Tr. Time (s)</td>
</tr>
<tr>
<td>Euclidean</td>
<td>10</td>
<td>0.09</td>
<td>12</td>
<td>0.80</td>
</tr>
<tr>
<td>Radon</td>
<td>31</td>
<td>0.22</td>
<td>12</td>
<td>0.15</td>
</tr>
<tr>
<td>Norm. Pix. Den.</td>
<td>43</td>
<td>0.30</td>
<td>13</td>
<td>0.16</td>
</tr>
<tr>
<td>Wav. Approx.</td>
<td>30</td>
<td>0.2</td>
<td>15</td>
<td>0.31</td>
</tr>
<tr>
<td>Modified Wav App</td>
<td>30</td>
<td>0.22</td>
<td>13</td>
<td>0.17</td>
</tr>
</tbody>
</table>

The overall recognition rates with both the types of wavelet features and the neural network parameters according to Table 6.1 are given in Table 6.3. It is evident from Table 6.3 that modified wavelet approximation features give better results all other feature extraction techniques studied so far. The Levenberg-Marquardt algorithm not only requires least number of epochs for training the neural network, but also gives higher recognition rates as compared to the other training algorithms considered so far.
A Neural Network Based Handwritten Character Recognition for Marathi Script

Table 6.3 Recognition performance using different features

<table>
<thead>
<tr>
<th>Sr. no.</th>
<th>Feature extraction technique</th>
<th>Resize factor</th>
<th>Number of features</th>
<th>Recognition rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Radon features</td>
<td>16x16</td>
<td>81</td>
<td>91.10</td>
</tr>
<tr>
<td>2.</td>
<td>Euclidean features</td>
<td>16x16</td>
<td>32</td>
<td>94.14</td>
</tr>
<tr>
<td>3.</td>
<td>Normalized pixel density features</td>
<td>70x50</td>
<td>35</td>
<td>95.00</td>
</tr>
<tr>
<td>4.</td>
<td>Wavelet approximation features</td>
<td>16x16</td>
<td>64</td>
<td>95.68</td>
</tr>
<tr>
<td>5.</td>
<td>Modified wavelet approximation features</td>
<td>16x16</td>
<td>75</td>
<td>96.05</td>
</tr>
</tbody>
</table>

From Table 6.4 it is evident that LM performs better than GD, CG and QN algorithms. The performance of the features derived from wavelet transform is also studied in case of single stage system.

Table 6.4 Comparative study of performance of various learning algorithms

<table>
<thead>
<tr>
<th>Sr. no.</th>
<th>Feature extraction technique</th>
<th>Recognition rate (%) with GD</th>
<th>Recognition rate (%) with CG</th>
<th>Recognition rate (%) with QN</th>
<th>Recognition rate (%) with LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Radon features</td>
<td>89.00</td>
<td>89.45</td>
<td>89.62</td>
<td>91.10</td>
</tr>
<tr>
<td>2.</td>
<td>Euclidean features</td>
<td>90.14</td>
<td>91.34</td>
<td>92.12</td>
<td>94.14</td>
</tr>
<tr>
<td>3.</td>
<td>Normalized pixel density features</td>
<td>91.54</td>
<td>92.40</td>
<td>93.81</td>
<td>95.00</td>
</tr>
<tr>
<td>4.</td>
<td>Wavelet approximation features</td>
<td>93.00</td>
<td>93.90</td>
<td>94.79</td>
<td>95.68</td>
</tr>
<tr>
<td>5.</td>
<td>Modified wavelet approximation features</td>
<td>94.55</td>
<td>94.93</td>
<td>95.23</td>
<td>96.05</td>
</tr>
</tbody>
</table>
Table 6.5 presents a comparative study of these features with and without structural classification. Again, it is evident from these results that the structural classification improves the recognition accuracy.

**Table 6.5 Comparative study of performance of wavelet based features with and without structural classification**

<table>
<thead>
<tr>
<th>Sr. no.</th>
<th>Feature extraction technique</th>
<th>Resize factor</th>
<th>Number of features</th>
<th>Recognition rate (%)</th>
<th>with structural classification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Wavelet approximation features</td>
<td>16x16</td>
<td>64</td>
<td>87.42</td>
<td>95.68</td>
</tr>
<tr>
<td>2.</td>
<td>Modified wavelet approximation features</td>
<td>16x16</td>
<td>75</td>
<td>89.15</td>
<td>96.05</td>
</tr>
</tbody>
</table>

Thus the blocks in the proposed system are finalized after analyzing the results as indicated above. This block schematic of the proposed system is shown in Figure 6.19. The blocks in Figure 6.19 indicated the flow of the proposed system for Marathi handwritten characters. The system consists of two phases namely, training phase and testing phase. In the training phase, the character database is created. These characters are pre-processed and passed to structural classification stage to classify them into one of the 24 classes as per their structural features.

The modified wavelet approximation features are extracted from these characters to train the neural network for recognition. A neural network is built for each of the structural class with inputs equal to 75 and outputs equal to the number of characters in the respective structural class and one hidden layer. The number of hidden neurons is chosen to be equal to square root of the product of inputs and the outputs for the respective structural class. The activation function for the hidden layer is hyperbolic tangent sigmoid transfer function and linear for the output layer. The learning algorithm selected is Levenberg-Marquardt algorithm. The network created on selecting the mentioned parameters is then training using the features extracted and the final weights and biases are stored to be used during the testing phase.
In the testing phase, the candidate characters used for testing undergo the similar process till feature extraction. The neural network forward pass is calculated further using the features and the saved weights and the biases to obtain the final recognition results.

![System block schematic for Marathi character recognition](image)

Figure 6.19 Proposed system block schematic for Marathi character recognition

### 6.6 Concluding Remarks

Various steps towards performance optimization are taken. Two new wavelet based feature extraction methods are utilized. They are namely, wavelet approximation features and modified wavelet approximation features. The neural network parameters are changed and the optimum combination is obtained that gives the best results. Results
showed that the network with single hidden layer, with neurons equal to square root of the product of input and output, hyperbolic tangent sigmoid transfer function and Levenberg-Marquardt training algorithm proves to be the best (refer Table 6.4). The Levenberg-Marquardt training algorithm requires the least number of epochs for training the network as indicated in Table 6.2. Moreover this network with the selected parameters and the modified wavelet approximation features gives the highest recognition accuracy of 96.05% over other feature extraction techniques (Table 6.3). Also the multistage recognition system improves the results over single stage system by 8.26% and 6.89% in case of wavelet approximation features and modified wavelet approximation features respectively (refer Table 6.5). Since the multistage recognition gives higher recognition rates in all the features extractions techniques discussed so far, this type of system is used in the following chapters.